Chaos Engineering

Recitation 9, CMU 17-313, Fall 2020

Goal: During this recitation, we will learn to look at how chaos engineering style experiments can expose vulnerabilities in microservice application code.

Setup:

1. Clone and build Filibuster:

    ```
    git clone git@github.com:CMU-313/filibuster.git
    cd filibuster
    conda create -n recitation9 python=3.6 anaconda
    conda activate recitation9
    conda config --set pip_interop_enabled True
    pip install -r requirements.txt
    ```

    Note: If you already installed and cd'd to this directory, remember to activate the conda virtual environment.

Part 1 (10 minutes max):

2. Run the integration test for the users service:

    ```
    python -m pytest -vv -s --cov=tests/cinema -k test_cinema_user_user_bookings
    ```

3. Generate code coverage metrics and examine the code coverage metrics:

    ```
    coverage html
    ```

    Open the `htmlcov/tests_cinema_services_users_py.html` file in your browser.

4. Open the code coverage metrics reported in:

    ```
    htmlcov/tests_cinema_services_users_py.html
    ```

5. Based on the code coverage metrics reported, what are the types of failures that we might be worried about and would want to test further? Only concern yourself with the bookings method for the users service (Lines 42 – 85, `users.py`)

    For each failure, do the following:

    a. Identify the precise failure you are concerned with and the line number where you think the failure will occur.

    b. Hypothesize what will happen if this failure occurs. For example, what will the output of the bookings service be?

6. Examine the code for other types of failures that might happen that might not be captured by the code coverage metrics. Only concern yourself with the bookings method for the users service (Lines 42 – 85, `users.py`)

   For each failure, do the following:

   a. Identify the precise failure you are concerned with and the line number where you think the failure will occur.

   b. Hypothesize what will happen if this failure occurs. For example, what will the output of the bookings service be?

Part 2 (30 minutes max):

7. Run the Filibuster tool to perform chaos experiments:

   ```
   python -m pytest -vv -s -k test_cinema_user_user_faulty_bookings
   ```

   You will find a vulnerability.

   You will have to address two things based to address this vulnerability.

   a. Fix the code to prevent against the vulnerability.

   Hint: If you are having trouble finding the vulnerability, use the stacktrace and what we learned in code archaeology to localize the fault.

   b. Fix the test to compensate for your fix. If you returned an error to deal with the vulnerability, the test will have to be adjusted to pass when it sees that error.

   Hint: The test is only assuming everything is going to work by default.

   c. Does the test pass that first vulnerability?

   If so, there are two others! See if you can find them and fix them.

   d. When you're done, run the following to generate a new coverage report.

   ```
   python -m pytest -vv -s --cov=tests/cinema -k \
       test_cinema_user_user_faulty_bookings

   coverage html
   ```

   How does it look? Does the coverage metric improve?

   Were you able to predict these vulnerabilities?

Part 3 (10 minutes):

8. We will walk through fixing the first bug together!